SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1862

DODATAK ZA APLIKACIJU DISCORD NAMIJENJEN UČINKOVITOJ PRIPREMI SASTANAKA U RADNOM OKVIRU SCRUM

Ivan Vjekoslav Rođak

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1862

DODATAK ZA APLIKACIJU DISCORD NAMIJENJEN UČINKOVITOJ PRIPREMI SASTANAKA U RADNOM OKVIRU SCRUM

Ivan Vjekoslav Rođak

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

ZAVRŠNI ZADATAK br. 1862

Pristupnik:	**Ivan Vjekoslav Rođak (0036546602)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	izv. prof. dr. sc. Ivana Bosnić

Zadatak: **Dodatak za aplikaciju Discord namijenjen učinkovitoj pripremi sastanaka u radnom okviru Scrum**

Opis zadatka:

Razvojni timovi koji se temelje na agilnim metodama, posebno na radnom okviru Scrum, često imaju sastanke timova, poput dnevnih sastanaka (engl. daily standups), čija je namjena brzo upoznavanje s trenutnim stanjem projekta i koracima u bližoj budućnosti. Ovi timovi često koriste aplikacije za brzu suradnju i razmjenu informacija. U ovome radu potrebno je proučiti agilni način razvoja programske potpore, posebno radni okvir Scrum te njegove elemente za suradnju članova tima. Istražiti kolaboracijsku platformu Discord, njenu arhitekturu te način razvoja dodataka. Za aplikaciju Discord razviti dodatak namijenjen učinkovitoj pripremi sastanaka u okviru Scrum. Dodatak treba podržavati funkcionalnosti korisne za pojedini tip sastanka, npr. brzi unos planiranih i završenih zadataka, dodjelu zadataka članu tima te prijavu problema poput kašnjenja ili komunikacijskih problema u timu. Dodatak treba omogućiti jasan i strukturirani pregled dobivenih informacija svim članovima tima te prikaz statistike po određenim parametrima. Testirati i komentirati ostvareno rješenje.

Rok za predaju rada: 23. lipnja 2025.

Sadržaj

Uvod

Porastom složenosti računalnih sustava javila se potreba za učinkovitim i strukturiranim vođenjem programskih timova. Uz početne zahtjeve za učinkovit razvoj i dobar timski rad, kroz vrijeme sve bitnije postaje razviti programsko rješenje u što kraćem roku. Ovakvi zahtjevi prouzročili su nastanak agilnih metoda razvoja početkom 21. stoljeća. Najpopularniji i najpoznatiji agilni radni okvir, ali ujedno programerima i jedan od najmrskijih [5], jest radni okvir Scrum koji zahtjeva dnevne sastanke (*daily standups*). Afera između programera i sastanaka (ustvari bilo kojeg nepotrebnog kontakta s ljudima) traje još od začetaka ove struke. Dnevni sastanci, preveliko miješanje nadređenih i nadređeni koji ne znaju o čemu pričaju glavni su nedostatci okvira Scrum, iz perspektive samih programera, te bi oni žarko željeli da se vrijeme potrošeno na sastancima svede na najmanju moguću mjeru.

S druge strane, razvoj tehnologije doveo je do pojave novih kanala i sredstava komunikacije. Aplikacije za komunikaciju kao što su *Zoom, Microsoft Teams* i *Slack* djelomično su zamijenile kontakt s ljudima uživo, pogotovo za vrijeme i nakon pandemije koronavirusa. U zadnje vrijeme, jedna platforma postaje sve popularnija u krugovima za koje nije prvenstveno namijenjena, zahvaljujući svojoj dostupnosti, jednostavnosti korištenja i velikoj prilagodljivosti. Riječ je o platformi *Discord*, prvenstveno namijenjenoj g*amerima*, a koja je svoje primjene našla i u drugim krugovima. Može li se spojiti naizgled nespojivo: *gamerska* platforma i rad u *corporate* svijetu, s naglaskom na skraćivanje sastanaka i poboljšanje produktivnosti?

Odgovor na ovo pitanje pokušao se dati kroz ovaj rad. Prvo će se navesti različite modele procesa programskog inžinjerstva u prvom poglavlju, kako se iz nedostataka klasičnih metoda razvio Scrum, potom koja su rješenja za komunikaciju već razvijena, u drugom poglavlju. Zatim će biti opisana platforma *Discord* što će poslužiti kao uvod u srž ovog rada: *Discord bot* i pripadajući *dashboard*. Kratki pregled arhitekture sustava dan je u trećem poglavlju, potom se u četvrtom i petom opisuju implementacija i funkcionalnosti. Dat će se zatim osvrt na odrađeni posao u šestom poglavlju prije samog zaključka.

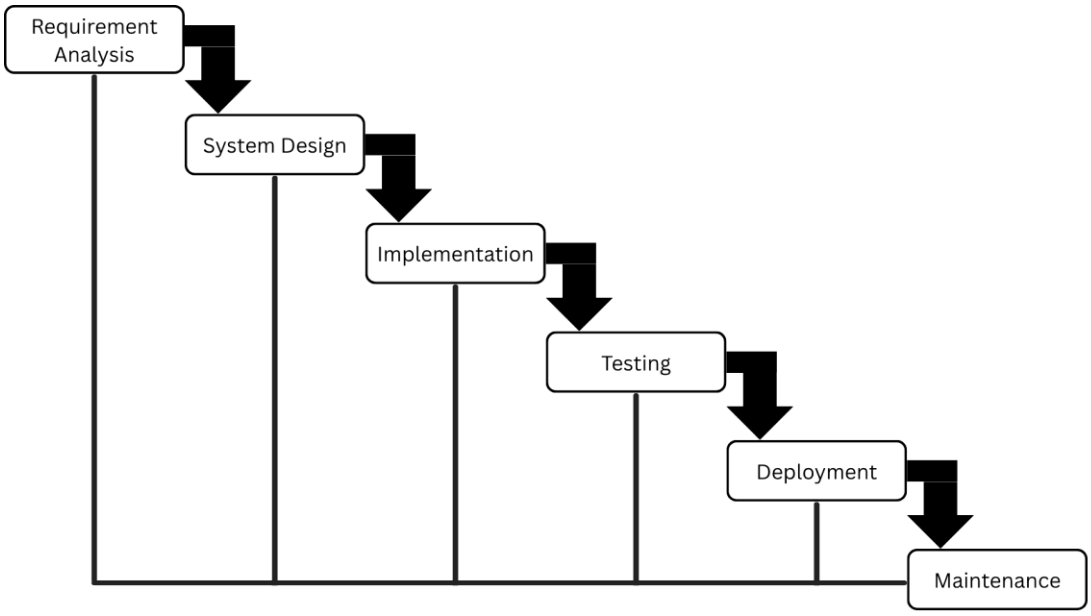
1. Modeli procesa programskog inženjerstva

Proces programskog inženjerstva definira se kao „strukturirani skup aktivnosti koji čini okvir neophodan za izvođenje sustavnog plana razvoja i oblikovanja programske potpore“ [1]. Neophodne aktivnosti za oblikovanje programske potpore su: specifikacija sustava, oblikovanje i implementacija, validacija i verifikacija, evolucija. Ove aktivnosti zajedničke su svim modelima, a razlike između modela svode se na raspored ovih aktivnosti unutar procesa. Model procesa programskog inženjerstva je apstraktna reprezentacija procesa, opisuje kako su aktivnosti vremenski raspoređene unutar samog procesa [1].

1.1. Tradicionalni modeli

Jedan od prvih modela, ujedno i najjednostavniji, jest *ad hoc* ili oportunistički model. U ovome se modelu odmah kreće na implementaciju, bez razrade zahtjeva, bez apstraktnog oblikovanja sustava, bez ocjene napretka, bez ičega. Programska se podrška mijenja dok se ne zadovolje korisnički zahtjevi. Glavni nedostatak ovoga modela prilično je jasan: nema nikakvog unaprijednog planiranja. Ovakav model prikladan je isključivo za najjednostavnije projekte, primjerice jednostavnija laboratorijska vježba na FER-u, te ga se zbog njegovih velikih mana neće dalje razmatrati [1].

Korak u pravom smjeru bio je vodopadni model. Opisao ga je William W. Royce 1970. godine, vodopadni je model postao temelj za daljnji razvoj modela. Aktivnosti su unutar vodopadnog modela raspoređene, kako mu samo ime kaže, poput vodopada: aktivnost ne može početi dok prijašnja aktivnost ne završi (Slika 1.1) [2].



Slika 1.1 Prikaz vodopadnog modela [2]

Glavne aktivnosti vodopadnog modela su sljedeće: analiza zahtjeva, oblikovanje sustava, implementacija i ispitivanje sustava, integracija i ispitivanje, rad i održavanje sustava [2]. Kako je već spomenuto, u vodopadnom modelu aktivnost ne može početi dok druga ne završi. Iz ovoga se jasno vidi da na početku projekta zahtjevi moraju biti jasno i precizno određeni prije negoli se može ići u oblikovanje sustava te potom u njegovu implementaciju. Ovo je ujedno i glavni nedostatak vodopadnog modela jer se zahtjevi gotovo uvijek mijenjaju tokom projekta. Vodopadni se model stoga može koristiti jedino u slučajevima kad su zahtjevi dobro poznati i stabilni za vrijeme trajanja projekta, a tehnologija implementacije mora biti dobro poznata i ne smije biti podložna promjenama [1].

Rješavanje problema rigidnosti vodopadnog modela dovelo je do razvoja inkrementalnog i iterativnog modela. Umjesto da se cijeli projekt radi odjednom, od početka do kraja, u inkrementalnom modelu on je razbijen u manje dijelove, inkremente. Početni se zahtjevi podijele po prioritetima te se prioritetniji zahtjevi implementiraju u ranijim inkrementima. Početkom svakog inkrementa zahtjevi se zamrznu te se eventualne promjene mogu realizirati tek u kasnijim inkrementima. Ovakvim pristupom korisnik dobiva nove funkcionalnosti svakim inkrementom, uz mogućnost eventualne korekcije zahtjeva. Rizik od neuspjeha također je manji nego kod drugih modela [1].

Iterativni model isporučuje proizvod tek na kraju samog projekta, no unutar samog projekta funkcionalnosti se ostvaruju u iteracijama. U svakoj iteraciji razvija se, testira i

validira dio programske potpore, i tako se ponavlja dok se projekt ne završi. Nakon svake iteracije, kupci proizvoda mogu dati povratnu informaciju koja će se uvažiti u sljedećoj iteraciji. Iterativni model fleksibilniji je od inkrementalnog, lako se prilagođava promjenama, pod cijenu nešto dužeg vremena razvoja [15].

Kao sljedeći korak u evoluciji modela može se spomenuti unificirani proces. Taj je model temeljen na obrascima upotrebe i UML-dijagramima, a cijeli se proces dijeli u inkremente. Naglašava se komunikacija s korisnikom i detaljna dokumentacija svih faza projekta, a u obzir se uzima i procjena rizika. Svaki inkrement podijeljen je na četiri faze: početak, razrada, izgradnja i prijenos. Svaka faza za sobom povlači dokumentaciju, popraćenu UML-dijagramima. Unificirani je proces detaljno razrađen te je vrlo pogodan za poslovna okruženja velikih tvrtki [1].

1.2. Agilni modeli

Koliko god tradicionalni modeli bili korisni i pogodni za korištenje u poslovnom svijetu, svi imaju jedan veliki nedostatak: administrativni teret i naglasak na poslovnu stranu, umjesto programiranje. Uz to, tradicionalni su se modeli pokazali vrlo krutima i nepogodnima za projekte s nestabilnim zahtjevima, ili za projekte koji zahtjevaju brzu isporuku inkremenata. Rješenje za ove probleme? Smanjiti administrativni teret, npr. smanjenjem količine potrebne dokumentacije, i naglasiti komunikaciju s kupcima [1].

Ovo su temelji agilnog ili ubrzanog razvoja (*agile development*). Agilni razvoj je naziv za grupu metoda kojima je zajednički razvoj uz male inkremente i brzi odziv na korisničke zahtjeve. U današnjem užurbanom svijetu, gdje je potrebno brzo razviti proizvod, agilne su se metode pokazale iznimno prikladnima [1].

Naglasak se u agilnim metodama stavlja na komunikaciju: komunikacija unutar programskog tima i komunikacija s kupcem. Kako su agilne metode modeli koji rade s promjenjivim zahtjevima te im se prilagođavaju, komunikacija s kupcem iznimno je bitna. Promjene zahtjeva uzrokuju i potrebu za brzom i čestom komunikacijom s programskim timom, stoga nije neuobičajeno vidjeti modele s vrlo čestim sastancima. Razvoj tehnologije doveo je do toga da se sastanci sve češće izvode virtualno, bez potrebe za kontaktom uživo, i time dodatno skraćuju vrijeme razvoja [1].

Radnih okvira agilnog razvoja ima mnogo, svaki sa svojim nijansama, načinima rada, prilagođeni za određene timove i specifične projekte. Najpoznatiji i najrašireniji radni okvir agilnog razvoja jest *Scrum*.

1.3. Scrum

Scrum je jedan od radnih okvira unutar agilnih metoda, ujedno i najpoznatiji i najrašireniji. Koristi iterativni, inkrementalni pristup uz kontrolu rizika. Jedna iteracija u Scrumu naziva se *Sprint*. *Sprint* traje između dva i četiri tjedna te mora na kraju dati neki mjerljivi rezultat. Glavni elementi Sprinta su: sastanak planiranja (*sprint planning*), posao razvoja s dnevnim sastancima (*daily standups*), revizija *Sprinta* (*Sprint review*) i reterospektiva *Sprinta* (*Sprint retrospective*) [1].

Tim u *Scrumu* sastoji se od pet do devet osoba, a mora sadržavati vlasnika proizvoda, vođu *Scruma* i razvojni tim. Vođa *Scruma*, kako mu ime nalaže, vodi cijeli tim i osigurava nesmetano provođenje procesa. Uz to, on je veza između vlasnika proizvoda i razvojnog tima te organizira i vodi sastanke [4].

Vlasnik je proizvoda odgovoran za jedan od artefakata *Scruma*: dnevnik zaostataka (*product backlog*). Dnevnik zaostataka jest sortirana lista svih zahtjeva na proizvod, a o prioritetima, sadržaju i redoslijedu odrađivanja odlučuje vlasnik proizvoda [4].

Ključni dio tima u svakom modelu, pa tako i u *Scrumu*, jest razvojni tim. Oni su ti koji stvaraju, implementiraju i razvijaju programsko rješenje te ga u konačnici i isporučuju. U *Scrumu*, oni sami biraju svoje zadatke iz dnevnika zaostataka, tvore jedan samoorganizirajući tim. Oni također sudjeluju u stvaranju dnevnika *Sprinta*, daju svoju procjenu koje će funkcionalnosti biti razvijene u sljedećem Sprintu te što je potrebno napraviti za njihovu realizaciju. Dnevnik *Sprinta* definira se na početku sprinta te sadrži detaljan plan kako bi se na dnevnim sastancima mogle razumjeti aktualne promjene [4].

Koliko god *Scrum* bio pametno osmišljen, uzimajući u obzir brzinu isporuke, smanjenje administrativnog tereta i naglašenu komunikaciju, on još uvijek ima svoje nedostatke. Planiranje i retrospektiva sprinta, i pogotovo dnevni sastanci, često predstavljaju nepotrebno gubljenje vremena, pogotovo za programere. Programeri ne vole sastanke te bi ih vrlo rado sveli na najmanju moguću mjeru, pogotovo kada sastanke vode ljudi koji nisu inženjeri.

2. Platforme za komunikaciju

Kako bi se ubrzala i poboljšala komunikacija, bilo unutar tima, bilo između pojedinih osoba, razvile su se više ili manje specijalizirane platforme. Još uvijek najrašireniji komunikacijski kanal, ujedno i jedan od prvih, jest elektronička pošta, no ona nije prikladna za održavanje sastanaka ni brze odgovore.

2.1. Klasične platforme

Platforme za komunikaciju moraju podržavati održavanje sastanaka u nekom obliku. Ovo se najčešće svodi na običan poziv s više ljudi. Eventualno se može u to uključiti i video pa se sastanak odvija preko videopoziva, vjernije simulirajući sastanak uživo, no to u većini slučajeva nije nužno. Uz mogućnost održavanja sastanaka, poželjno je na istoj platformi imati i mogućnost slanja tekstualnih poruka i datoteka.

2.1.1. WhatsApp

WhatsApp su razvili bivši zaposlenici tvrtke *Yahoo!* 2009. godine te je brzo ovladao svijetom [8]. O njegovoj popularnosti i znakovitosti svjedoči i činjenica da ga je kupila tvrtka *Facebook* 2014., kojoj je to tada bila najskuplja kupnja ikad [7]. Omogućuje slanje teksta, zvuka, slika, videa i datoteka preko interneta, dok se korisnički račun veže za broj mobitela. Podržava grupe i grupne pozive, sa i bez slike. Uz mobilnu verziju, postoji i inačica za osobna računala, no pristup njoj moguć je jedino uz korisnički račun na mobitelu i broj mobitela [9].

WhatsApp se primarno koristi za neformalnu komunikaciju između prijatelja i kolega, no nije posve prikladan za upotrebu u poslovnom svijetu. Ne postoji mogućnost zadavanja zadataka, grupe se vrlo lako preplave porukama te se tako bitne informacije vrlo lako izgube. Uz to, povezanost računa s osobnim brojem mobitela onemogućuje odvajanje privatnog i poslovnog života, osim za pojedince s više mobitela ili *dual SIM*-om [9].

2.1.2. Zoom

Uz *Twitterov Skype*, *Zoom* je najpoznatija i najpopularnija platforma za video pozive. Nastao je 2011. godine, ali je eksploziju popularnosti doživio tek u 2020. za vrijeme pandemije koronavirusa. Kako su uredi bili zatvoreni, bilo je potrebno komunikaciju i sastanke preseliti u virtualnu domenu. *Zoom* je ovdje kapitalizirao, dobivši 2.13 milijuna preuzimanja samo u ožujku, a ta se brojka popela do 500 milijuna do kraja godine. Nakon ublažavanja mjera, *Zoom* je još uvijek ostao vrlo popularan jer su ljudi otkrili brojne prednosti rada na daljinu (*remote work*) [10].

U besplatnoj varijanti, *Zoom* podržava do 100 korisnika i videopozive do 40 minuta. Najviša razina plaćene pretplate dolazi s podrškom za 1000 korisnika i videopozive do 30 sati [11].

Dok je *Zoom* s jedne strane optimiran za prijenos zvuka i slike, s druge mu strane nedostaje podrška za slanje poruka izvan samih videopoziva. *Zoom* je dobar odabir ako se traži samo zamjena za sastanke uživo, ali se ne može koristiti za općenitu komunikaciju unutar timova.

2.1.3. Microsoft Teams

Microsoft Teams razvijen je kao konkurencija *Slacku* i kao zamjena za *Skype for Business* i *Microsoft Classroom* [12]. Kao i *Zoom*, eksploziju popularnosti doživio je za vrijeme pandemije koronavirusa te se također zadržao i do danas. Nudi podršku za video i audiopozive, ali za razliku od *Zooma* nudi i prostor za slanje tekstualnih poruka i datoteka. Poruke se mogu slati direktno korisnicima ili pak u grupe (*Teams*) koje se mogu dodatno podijeliti u kanale. Također, nudi i prostor za spremanje datoteka čija veličina ovisi o kupljenoj verziji *Teamsa* [12].

Stvaranje korisničkog računa provodi se preko email adrese, za razliku od *WhatsAppa*. Također, u ponudi su aplikacije za računala, mobitele i druge prijenosne uređaje, te inačica za *web* preglednike. Uz mogućnost integracije s postojećim *Microsoftovim* proizvodima i s podrškom za vanjske aplikacije i servise, *Microsoft Teams* jedno je od najboljih rješenja za komunikaciju u poslovnom svijetu.

2.1.4. Slack

Slack je razvijen 2013. godine prvenstveno kao platforma za komunikaciju timova u poslovnom svijetu. Prije puštanja na opće tržište, *Slack* je razvijen i služio unutar tvrtke *Tiny Speck* za vrijeme razvoja igre *Glitch* [14]. Nudi podršku za grupe (*chat rooms*), organizirane po temama, privatne grupe i direktne poruke korisnicima. Također, podržava video i audio pozive uz mogućnost dijeljenja ekrana te slanje datoteka. Sve od navedenoga može se tražiti pomoću tražilice unutar *Slacka*. Povezanost s vanjskim servisima također je podržana, uz mogućnost razvoja vlastitih servisa i *chatbotova*. Broj korisnika po grupi i kanalu u teoriji je neograničen [13].

Postoji besplatna verzija *Slacka*, ali ona uvodi ograničenja poput traženja poruka maksimalno 90 dana unatrag [13]. Dok za tvrtke plaćanje za bolje verzije nije problem, besplatna platforma sa sličnim značajkama bila bi bolje rješenje.

2.2. Discord

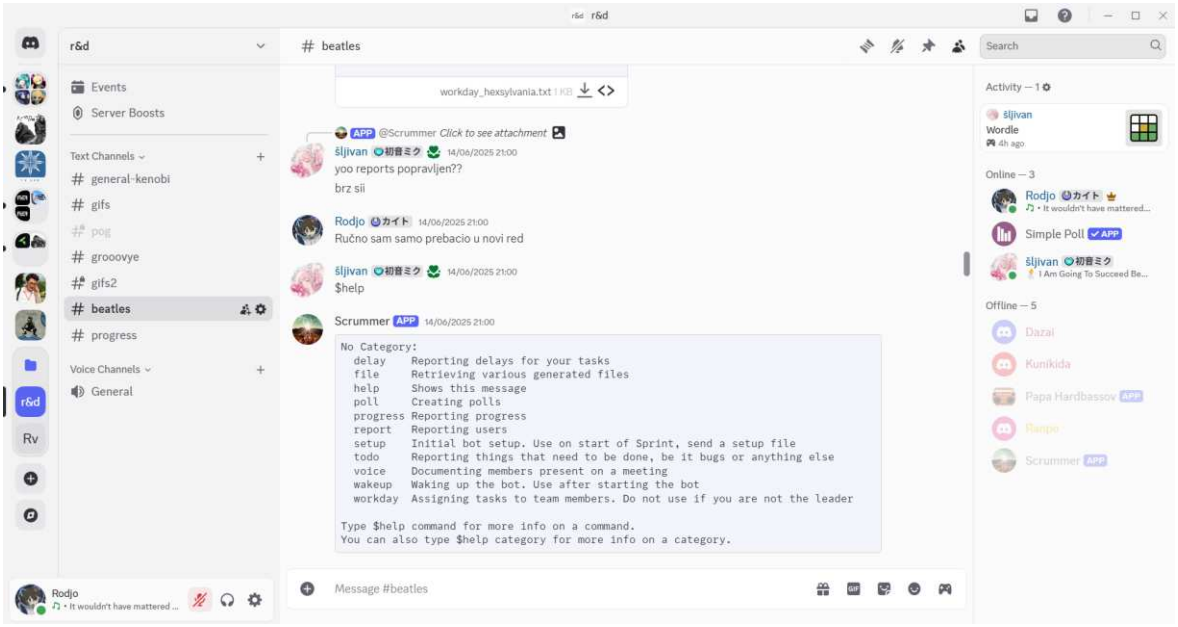
2.2.1. Što je Discord?

Početak *Discorda* sličan je početcima *Slacka*. Kreatori *Discorda* radili su u *gaming* studiju te su se žalili na već postojeće VoIP platforme. To iskustvo potaknulo ih je na razvoj platforme koja će biti usmjerena na lakoću korištenja i na minimalan utjecaj na performanse sustava [6].

Pušten u upotrebu kao *web* aplikacija 2015., u početku nije reklamiran za specifičnu publiku, no putem društvene mreže *Reddit Discord* je zaživio u *gaming* krugovima gdje je i do danas gotovo jedina platforma komunikacije. S vremenom je razvijena *desktop* aplikacija, uz verzije za *Android* i *iOS* [6].

Uz audiopozive, *Discord* podržava i tekstualne poruke, bilo u grupe (zvani *serveri*), bilo direktno korisnicima. Podržava također slanje datoteka, koje su u besplatnoj verziji ograničene na 10 MB [6].

Na slici je prikazan izgled *Discorda* (Slika 2.1). Na lijevoj strani nalazi se lista svih *servera* kojima se korisnik pridružio. Zatim, popis kanala u trenutnom *serveru*. S desne strane nalazi se popis svih članova *servera*. Konačno, u sredini se nalazi sam razgovor, s poljem za upis poruka.



Slika 2.1: Izgled *Discorda*

2.2.2. Primjene

Kao što je već spomenuto, glavna upotreba *Discorda* jest u *gaming* krugovima, za komunikaciju tijekom igranja i općenito čavrljanje. *Discord* funkcionira na temelju *servera* kojima se pristupa preko linka. Unutar *servera* moguće je definirati jedan ili više kanala (*channels*) te ih prilagoditi određenim temama. Uz to, moguće je svakom korisniku definirati ulogu (*role*) unutar *servera* te preko toga ograničiti pristup određenim kanalima i/ili funkcionalnostima.

Uz ove načine prilagodbe, glavni način prilagodbe *servera* različitim potrebama sastoji se od razvijanja vanjskih servisa, tzv. *botovi*, o kojima će riječi biti nešto kasnije. Ovako se *Discord server* može prilagoditi praktički svakom obliku upotrebe.

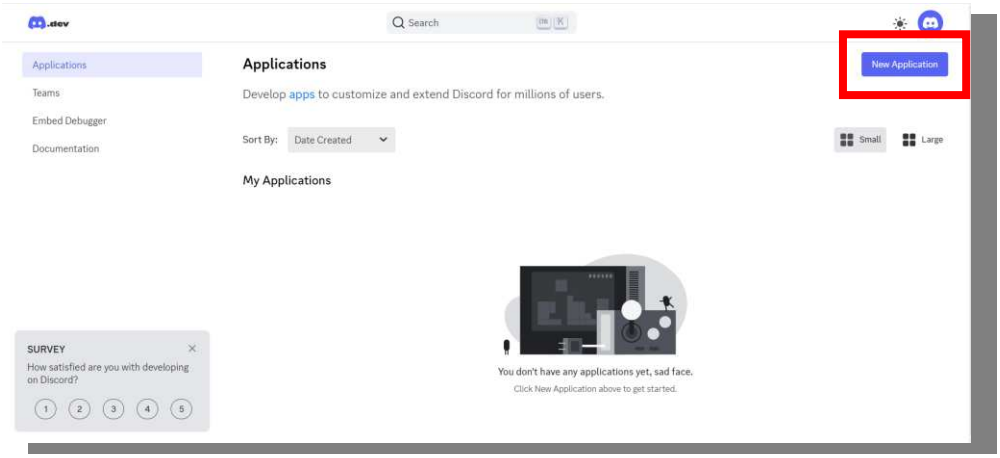
Osim za *gaming*, *Discord* se vrlo često koristi kao općeniti *chatroom* za opušteno čavrljanje, slušanje glazbe, gledanje filmova, dijeljenje umjetničkih djela itd. Vrlo su česti *serveri* koji se vežu za pojedine specifične interese (glazba, sport, popularni bendovi, popularni filmovi i serije, vlakovi, anime, *vocaloid*, itd.).

Korisnost, praktičnost, beskonačna mogućnost prilagodbe i činjenica da je *Discord* besplatan prepoznati su i u profesionalnom svijetu. Vjerojatno najpoznatiji primjer upotrebe *Discorda* u profesionalnom svijetu jest njegova upotreba u utrci *24 sata Le Mansa*, gdje sudci i direktori utrke komuniciraju preko *Discord servera*, u okruženju u kojemu je presudno brzo i pouzdano dijeljenje informacija te brzo donošenje odluka.

2.2.3. Discord API

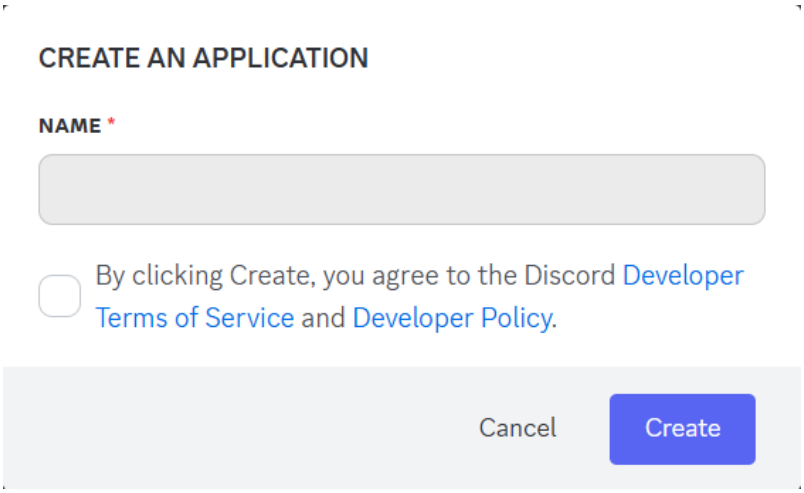
Discord nudi službeno programsko sučelje koje se koristi kako bi se *botovi* spojili sa *serverom* i kako bi se oni programirali. Programske biblioteke razvijene su za više programskih jezika, od kojih su najpopularniji *Python* i *Node.js* [3].

Kako bi se mogao stvoriti *bot token*, potrebno je ulogirati se svojim *Discord* korisničkim računom na *Discord Developer Portal* te stvoriti novu aplikaciju (Slika 2.2).



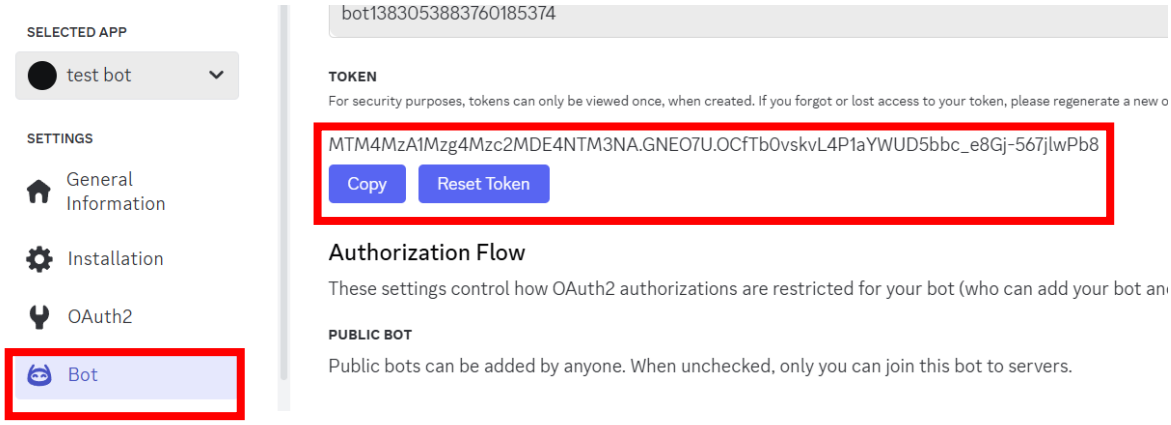
Slika 2.2: Discord Developer Portal

Klikom na *New Application* otvorit će se izbornik kao na slici (Slika 2.3).



Slika 2.3: Stvaranje novog *bota*

Potrebno je upisati željeno ime *bota* koje će se prikazati na *Discord serveru*. Zatim, potrebno je otići na podizbornik *Bot* i kopirati generirani *Bot token* (Slika 2.4).



Slika 2.4: Generiranje *bot tokena*

Iz sigurnosnih razloga, *bot token* se prikazuje samo jednom. U slučaju da programer zaboravi koji je *bot token* koristio, mora ga preko ove iste stranice resetirati. Generirani bot token potrebno je dodati u napisani program:

```
TOKEN = #your token here
bot.run(TOKEN)
```

Napomena: nije preporučljivo kôd s upisanim *bot tokenom* stavljati na GitHub. *Bot token* je jedina veza između programskog kôda i *Discord servera* tako da eventualni napadač može preko ukradenog *tokena* napraviti veliku štetu u *serveru* u kojemu se *bot* nalazi.

Kada je ovo napravljeno, ostatak se prepušta mašti programera. Svaka naredba *botu* definira se kao zasebna funkcija, postavi se dekorator, a sama funkcionalnost unutar te funkcije može biti što god osoba poželi.

3. Implementacija alata Scrummer

Ovim radom nastojao se dati odgovor na pitanje dano u uvodu: „Može li se spojiti naizgled nespojivo: *gamerska* platforma i rad u *corporate* svijetu, s naglaskom na skraćivanje sastanaka i poboljšanje produktivnosti?“ U tu svrhu odabrana je platforma *Discord* zbog činjenice da je besplatna, pruža službeno programsko sučelje za razvoj dodataka (u daljnjem tekstu: *bot*) te je izuzetno jednostavan za korištenje. Za pregledan prikaz podataka odlučeno je razviti *dashboard* u alatu *Flutter.* Zašto baš *Flutter*? Kôd napisan u alatu *Flutter* može se prevesti za izvođenje na nekoliko platformi: *web* inačica, *Windows, Android, Linux* i *MacOS*. Uz to, programski jezik *Dart* u kojemu se piše unutar *Fluttera* vrlo je pregledan, logičan i jednostavan za *debugging*, čineći tako odličnu alternativu drugim alatima za razvoj grafičkog sučelja, npr. *Reactu*.

3.1. Arhitektura

Alat *Scrummer* realiziran je u dva dijela: *Discord bot,* napisan u programskom jeziku *Python* pomoću kojega se prikupljaju podatci za vrijeme projekta i spremaju u tekstualne datoteke, i *dashboarda* razvijenog u programskom jeziku *Flutter* pomoću kojega se na pregledan način prikazuju podatci spremljeni u tekstualne datoteke.

Python skripta predviđena je za lokalno pokretanje, idealno s računala vođe tima, na čije će se računalo spremati i generirane datoteke. *Bot* se može koristiti samo dok je *Python* skripta pokrenuta, što ne bi trebalo predstavljati problem jer troši zanemarivo malo računalnih resursa. Pokrenut *bot* preko interneta komunicira sa *Discord serverom* i ostalim članovima tima.

Dashboard je razvijen kao zasebna aplikacija, a vezu s *Discord botom* ostvaruje isključivo preko generiranih datoteka. Također se pokreće lokalno, *offline*. Može ga pokrenuti bilo tko u timu, pod uvjetom da ima potrebne generirane datoteke koje mu se mogu poslati preko *Discord bota*.

Kako su *bot* i *dashboard* dvije odvojene aplikacije, one se mogu koristiti jedna bez druge, dokle god se poštuje format datoteka za *dashboard*. No, zbog formata datoteka, preporuča se koristiti ova dva dijela zajedno, kao jedan alat.

3.2. Funkcionalnosti

Discord bot sadržava sljedeće naredbe:

- *setup*
- *wakeup*
- *voice*
- *todo*
- *progress*
- *report*
- *file*
- *workday*
- *delay*

Implementacija i korištenje ovih naredbi bit će objašnjeni u narednim poglavljima.

Tijekom razvoja *bota* nastojalo se razviti funkcionalnosti korisne za poslovno okruženje te se također nastojalo same funkcionalnosti implementirati tako da se lako mogu izmijeniti, doraditi i prilagoditi. Također, kako je *bot* predviđen za lokalno pokretanje, njegovo se stanje sprema u posebnu datoteku. Veličina same skripte i vrijeme izvršavanja nisu predstavljali problem jer se radi o samo jednoj *Python* skripti koja samo parsira ulaz te ga sprema u odgovarajuću datoteku. Također, veličina generiranih datoteka se mjeri u kilobajtima.

Za pregledan prikaz skupljenih podataka, razvijen je *dashboard* u *Flutteru*. Podatci se mogu pregledavati na sljedećim stranicama:

- *General Information*
- *Progress*
- *Tasks & Delays*
- *Meetings*
- *Reports*

Što se točno prikazuje, i kako se prikazuje, bit će objašnjeno u narednim poglavljima. Svaka stranica funkcionira na sličan način: parsira se odgovarajuća datoteka, sortira se po članovima tima ako treba, te se prikaže na ekran.

4. Scrummer – Discord bot

U svrhu skraćivanja dnevnih sastanaka u *Scrumu* i poboljšanja komunikacije unutar tima, razvijen je *Discord bot* vrlo maštovitog imena *Scrummer*. Programsko rješenje implementirano je u programskom jeziku *Python* koristeći popularnu biblioteku *discord.py*.

4.1. Implementacija i funkcionalnosti

4.1.1. Početni kôd

Program napisan u *Pythonu* spaja se na *Discord* preko *bot tokena*, kako je to opisano u poglavlju 2.2.3 Kada se pokrene, on čeka na poruke koje počinju odgovarajućim prefiksom. U ovom slučaju, poruke koje će *bot* pročitati i obraditi moraju početi znakom „$“. Prefiks se definira u samom kôdu na sljedeći način:

```
bot = commands.Bot(command_prefix='$', intents=intents)
```

Ovime se ujedno stvorio objekt `bot` pomoću kojega se definiraju njegove naredbe. Uz prefiks, bitno je navesti i koje namjere (*intents*) *bot* ima. Za ispravan rad ovog *bota*, uz osnovne namjere koje su predefinirane, potrebno je navesti *intents* za čitanje poruka, članova *servera* i popisa *servera* u kojima se *bot* nalazi:

```
intents = discord.Intents.default()
intents.message_content = True
intents.members = True
intents.guilds = True
```

Također, u samom kôdu mora se navesti i direktorij gdje će se spremati generirane datoteke:

```
working_directory = "path to directory"
```

Broj naredbi koje se mogu definirati nije ograničen, a sve se u kôdu zadaju na sljedeći način:

```
@bot.command(name="command_name")
async def command_name(ctx, arguments)
```

Anotacijom `@bot.command(name=)` *botu* se u listu naredbi dodaje naredba imena koje se zadaje sa `name=` . Kada se prepozna naredba poslana na *Discord* (preko zadanog prefiksa), poziva se anotirana funkcija. `Ctx` je objekt koji sadrži podatke o kanalu, *serveru* i pošiljatelju poruke te se preko tog objekta može poslati poruka natrag.

4.1.2. Setup i wakeup

Scrummer ima implementirane naredbe prijave napretka na projektu, prijave zadataka, prijave kašnjenja na pojedinom zadatku, prijave članova *servera*, glasanje (*poll*), zadavanje i preuzimanje zadataka te zapisivanje sastanaka. Kako bi se pokrenuo rad *bota*, potrebno mu je poslati određene podatke preko naredbe `$setup`. Ova naredba uz sebe očekuje i .txt datoteku s informacijama o članovima tima, njihovim ulogama, datumu početka i završetka projekta, vremenu sastanaka, kanalu za slanje napretka i, opcionalno, razinama prioriteta. Ovi se podatci spremaju u klasu `Info` i u datoteku *setup_server_name.txt* kako bi se očuvalo stanje *bota.*

Kako je *bot* zamišljen tako da se ponovno pokrene svaki dan prije početka posla, definirana je i naredba $wakeup. Ona se može, ali i ne mora, pozvati eksplicitno jer se na početku svake druge naredbe poziva automatski ako već nije bila prije pozvana:

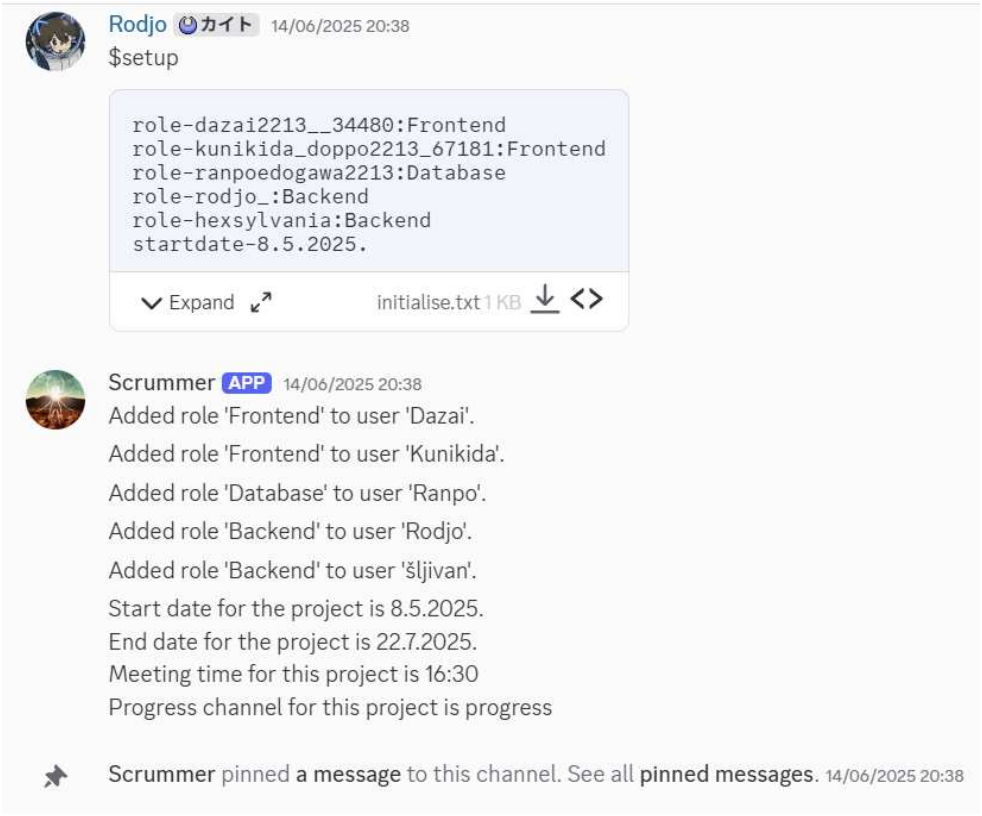
```
global setup_done
if not setup_done:
    await wakeup(ctx)
```

Ovom se naredbom automatski inicijalizira *bota* preko prethodno spremljene datoteke *setup_server_name.txt*. Datoteka ima sljedeći format:

```
server_name:<name>
member_count:<number>
startdate:<date, dd.mm.yyyy>
enddate:<date, dd.mm.yyyy >
meeting_time:<hh:mm>
progress_channel:<channel id>
current_week:<number>
priority:<number>
index:<number>
```

Trenutni tjedan prati se radi upisivanja u *progress.txt* datoteku o čemu će riječi biti nešto kasnije, kao i o *progress_channel* varijabli. *Priority* i *index* koriste se u *todo* naredbi.

Nakon uspješne inicijalizacije, *bot* šalje u *server* poruke o članovima tima i njihovim zaduženjima. Zatim, šalje poruku s osnovnim podatcima o projektu (početak, završetak, vrijeme sastanaka, kanal za *progress*) te tu poruku označi i prikvači (*pinned message*). Na slici je prikazan izgled tih poruka (Slika 4.1).



Slika 4.1: Izgled prve poruke *bota*

4.1.3. Todo

Jednom kada je *bot* inicijaliziran, može se krenuti s njegovim korištenjem. Za prijavu stvari koje je potrebno napraviti koristi se naredba $todo koja je implementirana kao sljedeća funkcija:

```
async def todo(ctx, priority: int, *, user_message: str)
```

Kao i sve druge funkcije preko kojih se obrađuju naredbe, i ova mora biti asinkrona. Ona prima dva argumenta: prioritet i korisnikovu poruku proizvoljne duljine. Što se veći broj preda kao prioritet, to taj zadatak postaje bitniji. Prioritet mora biti između 1 i broja koji je zadan prilikom inicijalizacije. Korisnikova poruka ustvari je opis posla koji se treba odraditi.

Za svaku *todo* poruku generira se novi *task id* tako da se *index* inkrementira te se osvježi njegova vrijednost u *setup* datoteci. *Task id* se tada zajedno s prioritetom i korisnikovom porukom sprema u datoteku *todo_<date>.txt* u sljedećem formatu:

```
<priority>,<task id>,<message>
```

Datoteka je sortirana po prioritetima: viši prioriteti nalaze se na vrhu datoteke. Za svaki se dan generira nova *todo* datoteka te se na kraju radnog dana, pola sata prije definiranog vremena sastanka, šalje korisniku koji je napravio *server*, za kojega pretpostavljamo da je vođa projekta. Uz zapisivanje u *todo* datoteku, prijavljeni zadatak zapisuje se i u *progress.txt* datoteku u sljedećem formatu:

```
<date> <time > | <username> | Reported task | <task> | <task id>
```

4.1.4. Workday

Task ids generirani ovom funkcijom koriste se za raspodjelu poslova članovima tima. Ova funkcionalnost implementirana je preko naredbe $workday i sljedeće funkcije:

```
async def workday(ctx, user: discord.User)
```

Ova funkcija prima objekt *user* kome će poslije biti poslani zadatci. Nakon poziva naredbe, *bot* skuplja poruke dok mu se ne pošalje *send*. Jedna poruka predstavlja jedan zadatak, a sve se poruke spremaju u datoteku *workday_<username>.txt* u sljedećem formatu:

```
Deadline,<date>,<task id>,<task description>
```

Kada se pošalje poruka *send*, ova se datoteka privatno pošalje korisniku predanom preko naredbe (*user*).

4.1.5. Voice

Na sličan način funkcionira i naredba $voice koja bilježi tko je prisustvovao sastanku te bilježi o čemu se na sastanku razgovaralo (*log*).

```
async def voice(ctx, channel_name: str)
```

Ovoj funkciji potrebno je predati naziv kanala u kojemu se odvija sastanak. Uz to, korisnik koji je pokrenuo tu naredbu mora i sam biti u tom kanalu.

Ova funkcija sada prikuplja poruke koje pristižu, sve dok se ne pošalje poruka *done*. Nakon toga, u datoteku *meeting.txt* zapisuju se bilješke sa sastanka, vrijeme početka i kraja sastanka i sudionici, u sljedećem formatu:

```
<date>,<time>,<voice channel name>,<list of users>
```

```
<log entry>
<log entry>
…
Meeting ended at <time>
```

4.1.6. Report

Ako neki član tima ne poštuje pravila ili ne ispunjava svoje obaveze, naredbom $report može ga se prijaviti.

```
async def report(ctx, user: discord.User, *, reason:str)
```

Funkcija prima korisnika (*user*) i razlog zašto ga se prijavljuje. Uz zapisivanje u datoteku *reports.txt*, poruka s korisnikom i razlogom prijave šalje se administratoru *servera*, u privatne poruke.

4.1.7. Progress

Prijavljivanje obavljenog posla odvija se preko naredbe $progress:

```
async def progress(ctx, id = 0, existing = 0, *, description:str)
```

Ako se prijavljuje završetak pojedinog zadatka, potrebno je poslati *task id* tog zadatka, a za parametar *existing* staviti nulu. Ako se prijavljuje samo napredak na nekom zadatku, potrebno je poslati *task id* i bilo koju pozitivnu vrijednost za parametar *existing*. U ovome slučaju, *existing* će označavati broj sati potrošenih na određeni zadatak te će se upisati u odgovarajuću *workday* datoteku, na mjesto gdje se nalazi zadatak zadan preko *task id.* Ako tamo još nije upisana vrijednost, upisat će se vrijednost *existing*, a ako postoji, *existing* će se pribrojiti postojećoj vrijednosti.

Ako se prijavljuje neka općenita odrađena stvar (npr. „Dogovorio sam sastanak s kupcem"), potrebno je kao *task id* i kao *existing* poslati nulu. Sve što se pošalje preko *progress* naredbe zapisuje se u *progress.txt* datoteku sa sljedećim formatom:

```
<date and time> | <user> | <description>
```

Ako se prijavljuje napredak na nekom zadatku, ovoj liniji bit će dodano:

```
Reported progress on <task id>
```

Ako se prijavljuje završeni zadatak, bit će napisano:

```
Resolved task number <task id>
```

Ova datoteka šalje se u poseban kanal (određen inicijalizacijskom datotekom) na kraju svakog radnog tjedna, sve do kraja projekta.

4.1.8. File

Sve navedene datoteke moguće je dohvatiti pomoću naredbe $file:

```
async def file(ctx, file, exact = 0)
```

Preko *file* može se poslati puno ime datoteke (npr. *meeting.txt*, uz postavljanje zastavice *exact* u bilo koju pozitivnu vrijednost) ili se može poslati ime datoteke bez ekstenzije i eventualnog dodatka imena korisnika, uz postavljanje vrijednosti *exact* u nulu. Za slučaj *todo* i *workday* datoteka, poslat će se sve *todo* i *workday* datoteke koje postoje u direktoriju, osim ako je navedeno potpuno i točno ime datoteke.

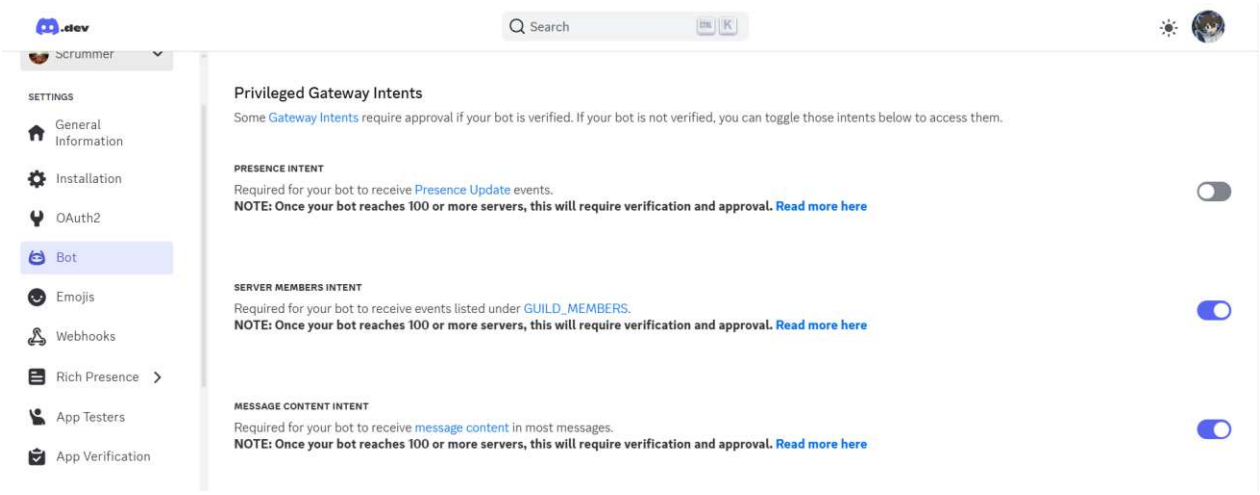
4.2. Inicijalizacija i korištenje

4.2.1. Kreiranje bot tokena i preuzimanje skripte

Preduvjet za korištenje *bota* jest preuzeta aplikacija *Discord* i stvoreni *server*. Unutar *servera* trebaju se nalaziti barem dva tekstualna kanala, jedan za općenito slanje poruka i jedan u koji će samo *bot* moći slati poruke, za slanje *progress.txt* datoteka. Preporuča se, radi preglednosti, stvoriti zaseban kanal za slanje naredbi *botu* (npr. za *progress, todo* i *report*) kao i još jedan kanal u koji će moći poruke slati samo privilegirani korisnici (npr. za *workday* i *meeting* naredbe). Ovo nije obavezno, ali se čestim slanjem naredbi *botu* vrlo lako može preplaviti glavni kanal za komunikaciju (najčešće zvan *General*). Uz tekstualne kanale, mora postojati i kanal za pozive (*voice channel*), no on se najčešće stvara automatski kad se stvori *server*. Uz kanale, moraju se definirati i uloge (*roles*) unutar *servera* kako bi ih se moglo raspodijeliti preko inicijalizacijske datoteke.

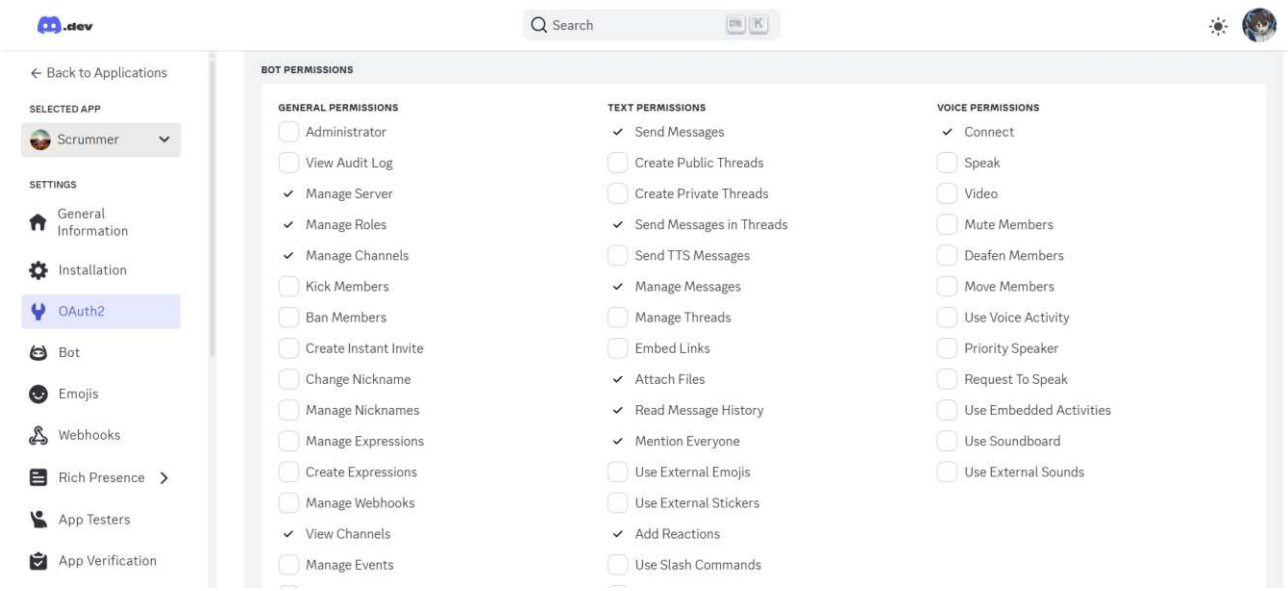
Program se preuzima s GitHub repozitorija. Program se može koristiti i u ovom obliku, a moguće ga je izmijeniti i prilagoditi vlastitim potrebama.

Nakon što je stvoren *bot token* na *Discord Developer Portalu* i nakon što je taj *token* zalijepljen u kôd, potrebno je još omogućiti *botu* određene privilegije. U podizborniku *bot* trebaju se označiti opcije *Public Bot, Server Members Intent* i *Message Content Intent* (Slika 4.2).



Slika 4.2: Omogućavanje *intents*

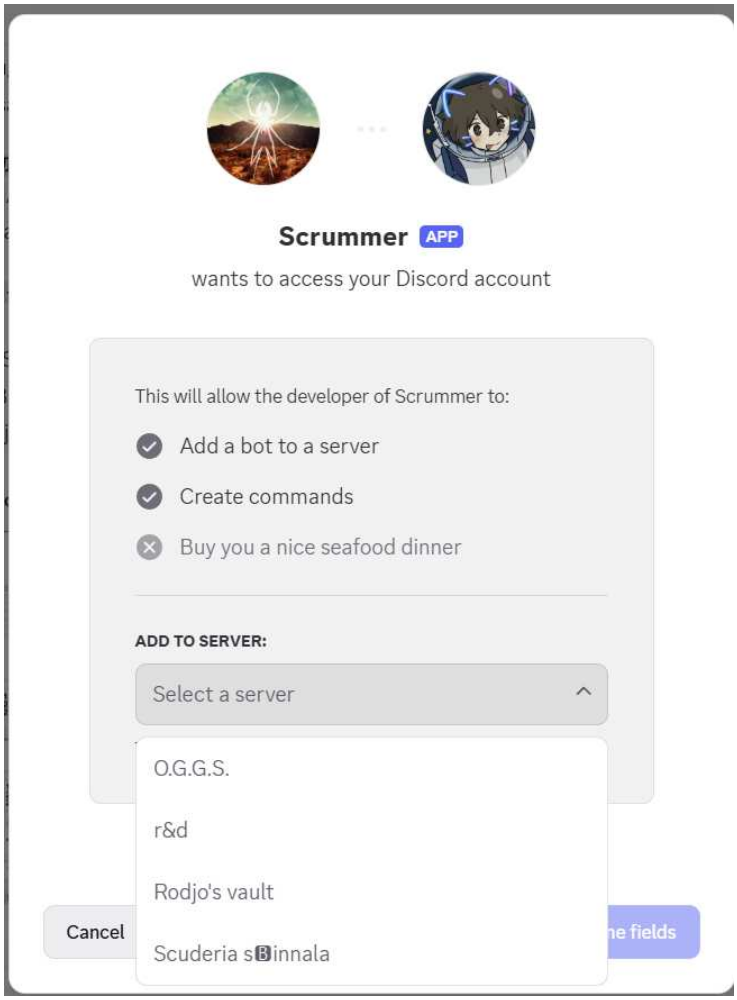
Sada je sve spremno za „pozivanje“ (*invite*) *bota* u željeni *server*. U izborniku se odabere kategorija *OAuth2* te se pod *scopes* odabere *bot.* Otvorit će se izbornik *Bot Permissions* u kojemu je potrebno označiti sljedeće: *Manage Server, Manage Roles, Manage Channels, View Channels, Moderate Members, Send Messages, Send Messages in Threads, Manage Messages, Attach Files, Read Message History, Mention Everyone, Add Reactions, Create Polls* i *Connect* (Slika 4.3).



Slika 4.3: *Bot Permissions*

Na dnu stranice pojavit će se generirani URL koji je potrebno kopirati i zalijepiti u *web* preglednik. Napomena: *bota* mogu „pozvati“ isključivo administratori i kreatori *servera*.

Kada se otvori URL, pojavit će se prozor kao na slici (Slika 4.4) u kojemu je potrebno odabrati željeni *server*.



Slika 4.4: Pozivanje *bota* u *server*

Sada kada je *bot* u *serveru*, potrebno je još samo pokrenuti *Python* skriptu. Skriptu pokreće jedna osoba u timu, s preporukom da to bude vođa tima. Svi članovi *servera* mogu koristiti naredbe *bota*.

4.2.2. Inicijalizacija - setup

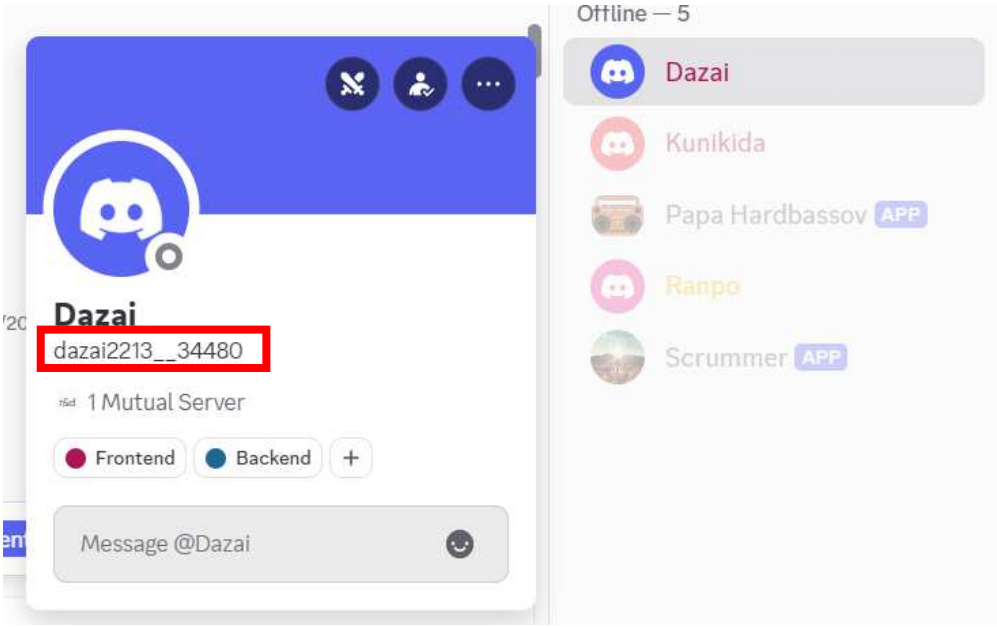
Prije samog početka rada, potrebno je provesti inicijalizaciju. Inicijalizacija se provodi slanjem poruke „$setup“ zajedno s datotekom koja se **mora** zvati *initialise.txt* te se nalaziti u direktoriju zadanom u kôdu, gdje se spremaju ostale datoteke. Ovo je potrebno kako bi *dashboard* ispravno funkcionirao.

Inicijalizacijska datoteka mora imati sljedeću strukturu:

```
role-<username1>:<rolename>
role-<username2>:<rolename>
…
startdate-<date>
```

```
enddate-<date>
meeting_time-<time>
progress_channel-<channel name>
priority-<priority levels, optional>
```

Redoslijed linija nije bitan, ali njihova struktura jest. *Username* se uzima s *Discorda* (Slika 4.5), a *rolename* su imena uloga koje moraju biti predefinirane u *serveru*.



Slika 4.5: Određivanje *usernamea*

Progress channel označava kanal u koji će se slati *progress.txt*, a dovoljno je navesti njegovo ime. *Priority* je opcionalni parametar koji označava broj razina prioriteta. Ako se ne navede, automatski se postavlja na 3.

Nakon uspješne inicijalizacije mogu se koristiti sve naredbe *bota.* Svaka naredba mora započinjati prefiksom „$“ kako bi *bot* prepoznao da se radi o naredbi za njega. Naredbe se mogu slati iz bilo kojeg tekstualnog kanala.

4.2.3. Prijava zadataka - todo

Kako bi se prijavio posao koji je potrebno odraditi, koristi se naredba *todo* sa sljedećom sintaksom:

```
$todo <description>
```

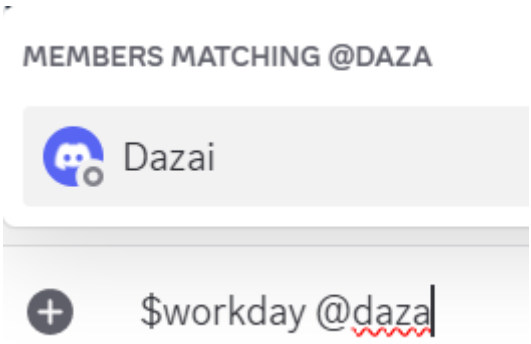
Description je u ovom slučaju kratak opis posla koji se treba odraditi.

4.2.4. Raspodjela zadataka - workday

Za zadavanje i preuzimanje poslova koristi se naredba *workday* sa sintaksom:

```
$workday @<username>
```

Username se predaje tako da se upiše „@“ i onda upiše prikazano ime korisnika (Slika 4.6).



Slika 4.6: Korisnikov @ *tag*

Bot sada prikuplja zadatke koji mu se šalju. Zadavanje zadataka ima sljedeću sintaksu:

```
<deadline> <task number> <task description>
```

Kao *deadline* se predaje 1 ako se zadatak treba odraditi do kraja tjedna ili 0 ako se zadatak trebao odraditi do kraja dana. *Task number* je *task id* zadatka ako on već postoji u nekoj *todo* datoteci, ili je 0 ako je novi zadatak. Kao *description* se predaje kratak opis posla.

4.2.5. Prijava kašnjenja - delay

Za prijavu kašnjenja potrebno je koristiti naredbu *delay*:

```
$delay <task id> <message>
```

Task id mora postojati i biti zadan članu tima kroz *workday* naredbu i datoteku. Član tima ne može prijaviti kašnjenje za zadatak koji mu nije zadan. Kao *message* se predaje kratko obrazloženje za kašnjenje.

4.2.6. Prijava napretka - progress

Kako bi se mogao pratiti napredak na projektu, implementirana je i naredba *progress*:

```
$progress <task id> <existing> <description>
```

Ako se želi prijaviti opći napredak (npr. „Dogovorio sam sastanak”), potrebno je kao *task id* i *existing* poslati 0, a u *description* poslati kratak opis posla.

Ako se prijavljuje završetak nekog zadatka, šalje se *task id* zadatka, a *existing* se postavlja u 0. Ovime se zadatak ujedno i označava kao završen.

Moguće je isto tako prijaviti potrošeno vrijeme na određenom zadatku. Potrebno je poslati *task id* i broj odrađenih sati u *existing*. Ovaj broj sati će se pribrojiti već zabilježenima, ako postoje.

4.2.7. Prijava članova tima - report

Za slučaj kada neki član tima ne poštuje pravila ili ne poštuje rokove, moguće ga je prijaviti administratoru *servera* pomoću naredbe *report*:

```
$report @<username> <reason>
```

Username se šalje na način opisan ranije (Slika 4.6), a kao *reason* se šalje obrazloženje prijave. Nakon pokretanja ove naredbe, administratoru *servera* se šalje privatna poruka, a član tima se zajedno s obrazloženjem sprema u posebnu datoteku.

Implementirana je također naredba pomoću koje se može provoditi glasanje:

```
$poll "<question>" "<option 1>" "<option 2>" … "<option 9>"
```

Ovime se stvara poruka na koju članovi tima mogu reagirati ponuđenim osjećajnicima koji su mapirani na određene odgovore. Ova naredba podržava do 9 opcija odgovora. Napomena: pitanje i svaki odgovor moraju zasebno biti stavljeni pod navodnike kako bi naredba ispravno funkcionirala.

4.2.8. Zapisnik sastanaka - voice

Za potrebe zapisivanja na sastancima implementirana je naredba *voice*:

```
$voice <channel name>
```

Kao *channel name* se mora predati ime kanala u kojemu se trenutno odvija sastanak. Također, osoba koja šalje ovu naredbu mora biti u tom kanalu.

Nakon pokretanja naredbe, *bot* zapisuje sve što osoba pošalje u taj kanal. Ovdje je potrebno pripaziti da se šalje samo ono što se želi zapisati u dnevnik sastanka (*meeting log*). Kada sastanak završi, potrebno je poslati poruku *done* čime se prekida prikupljanje stavaka.

5. Scrummer – dashboard

Preko *Discord bota* uspješno je implementirano skupljanje bitnih podataka o projektu i njihovo spremanje u tekstualne datoteke. Međutim, uz samu količinu tih datoteka, jasno je vidljivo da one same po sebi nisu pregledne te da su dosta nezgrapne za korištenje same po sebi. Javila se potreba za razvojem dodatka koji će te informacije prikazati na pregledniji način. U tu svrhu razvijen je ovaj *dashboard*.

5.1. Implementacija i funkcionalnosti

Dashboard je napisan u alatu *Flutter* koji se temelji na programskom jeziku *Dart*. Specifičnost *Fluttera* jest da se isti kôd može prevesti za više platformi. Tako se i ovaj *dashboard* može prevesti za upotrebu na *Windowsima*, *MacOSu*, *Linuxu*, *Androidu* ili kao *web* aplikacija, iako je predviđen za pokretanje na *Windowsima*, na osobnom računalu.

5.1.1. Općenita struktura dashboarda

Dasboard je implementiran kao skup stranica između kojih se bira pomoću *drop down* izbornika na glavnoj stranici. Svaka stranica implementirana je u zasebnoj datoteci radi modularnosti i lakšeg održavanja kôda. Svaka stranica sastoji se od klase koja sadržava podatke bitne za tu stranicu i klase koja modelira tu stranicu, obavezno sa `Widget build()` metodom koja je zadužena za renderiranje same stranice. Uz ovo, stranica može imati i dodatne klase i metode koji pomažu s obradom i prikazom informacija. Zbog učitavanja podataka iz datoteka, sve stranice implementirane su kao `StatefulWidget`. Način programiranja u *Flutteru* vrlo je sličan programiranju u radnom okviru React, uz veću jasnoću i preglednost kôda.

Datoteka *main.dart* preusmjeri program na prikaz glavne stranice (*homepage*):

```
@override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Scrummer',
      home: HomePage(),
    );
```

```
  }
```

5.1.2. Glavna stranica

Glavna stranica tražit će od korisnika da unese putanju do direktorija s datotekama koje je generirao *bot*. Kako ne bi korisnik trebao kôd svakog pokretanja ponovno unositi putanju, implementirano je spremanje i automatsko učitavanje te putanje:

```
Future<void> loadSavedFolderPath() async {
    final prefs = await SharedPreferences.getInstance();
    final savedPath = prefs.getString('savedFolderPath');
    if (savedPath != null) {
      setState(() {
        folderPath = savedPath;
      });
    }
  }
```

Ako putanja postoji, spremit će se u klasu `_HomePageState` koja modelira glavnu stranicu. Ako ne, varijabla će ostati prazna te će korisnik morati ručno odabrati putanju.

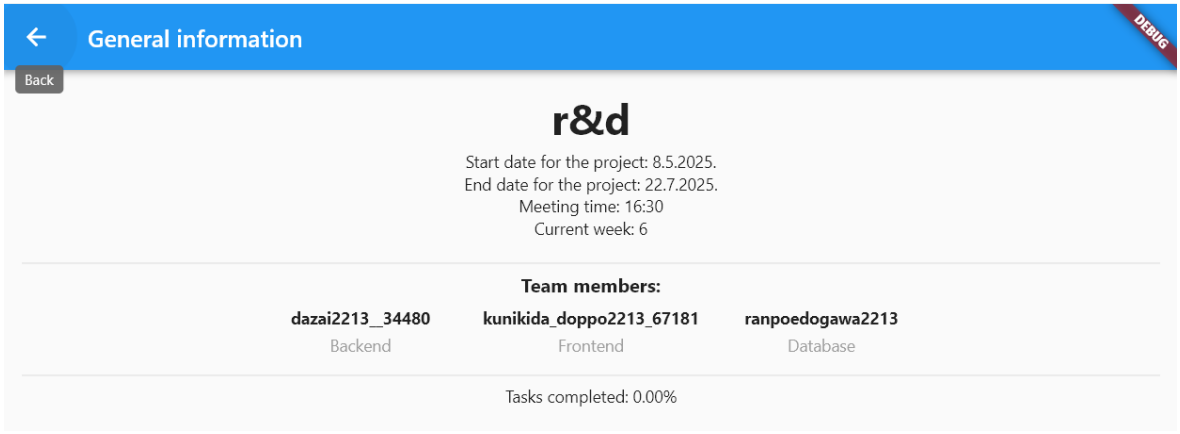
Nakon učitavanja putanje, program u zadanom direktoriju traži datoteku *initialise.txt* iz koje će iščitati članove tima i njihove uloge:

```
Future<void> loadUsers() async{

    final file = File('$folderPath/initialise.txt');
    final lines = await file.readAsLines();
    final Map<String, String> loadedUsers = {};

    for(var line in lines){
      final parts = line.split("-");

      if(parts[0] == "role"){
        final user = parts[1].split(":");
        loadedUsers[user[0].trim()] = user[1].trim();
      }
    }

    setState(() {
      users = loadedUsers;
    });
  }
```

Ako direktorij ne sadrži *initialise* i *setup* datoteke, *dashboard* se neće moći pokrenuti te će se ispisati upozorenje. Mapa *users* proslijedit će se poslije svakoj stranici jer svaka stranica prikazuje podatke sortirane po članovima tima. Kako bi se izbjeglo ponovno učitavanje članova iz datoteke svaki put kad se otvori neka stranica, učitavanje se izvodi samo jednom. na glavnoj stranici.

5.1.3. Prikaz osnovnih informacija - General information

Pomoću *dropdown* izbornika bira se željena stranica te se potom preusmjeruje na nju. Stranica *generalinfo* prikazuje osnovne informacije o projektu iz *initialise* i *setup* datoteka.

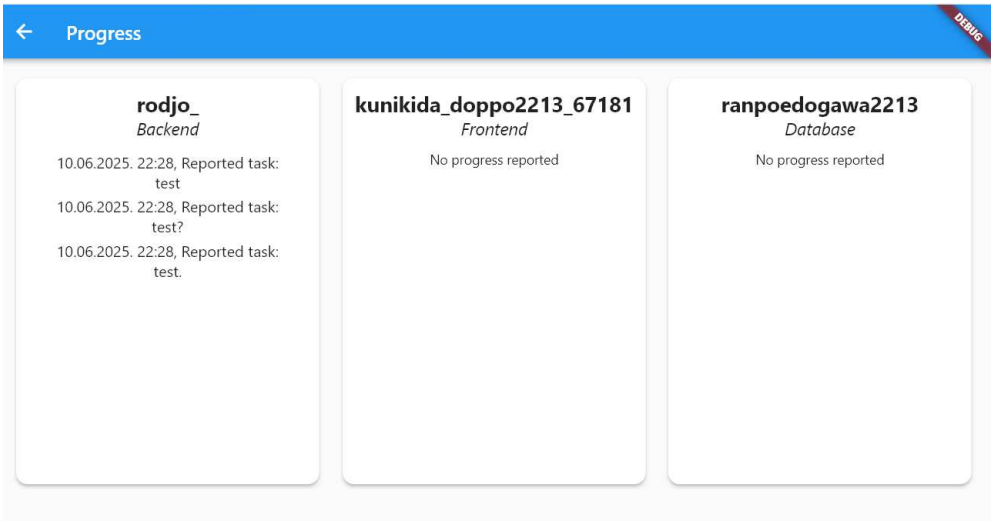


Slika 5.1: Stranica *General Information*

Prikazuju se ime *servera*, datum početka i završetka projekta, vrijeme sastanaka, trenutni tjedan, popis članova tima i njihovih uloga i postotak zadataka označenih kao završeni. Kao imena članova tima uzimaju se njihovi *Discord usernames*, kako su navedeni u datoteci *initialise.txt*.

5.1.4. Prikaz napretka - Progress

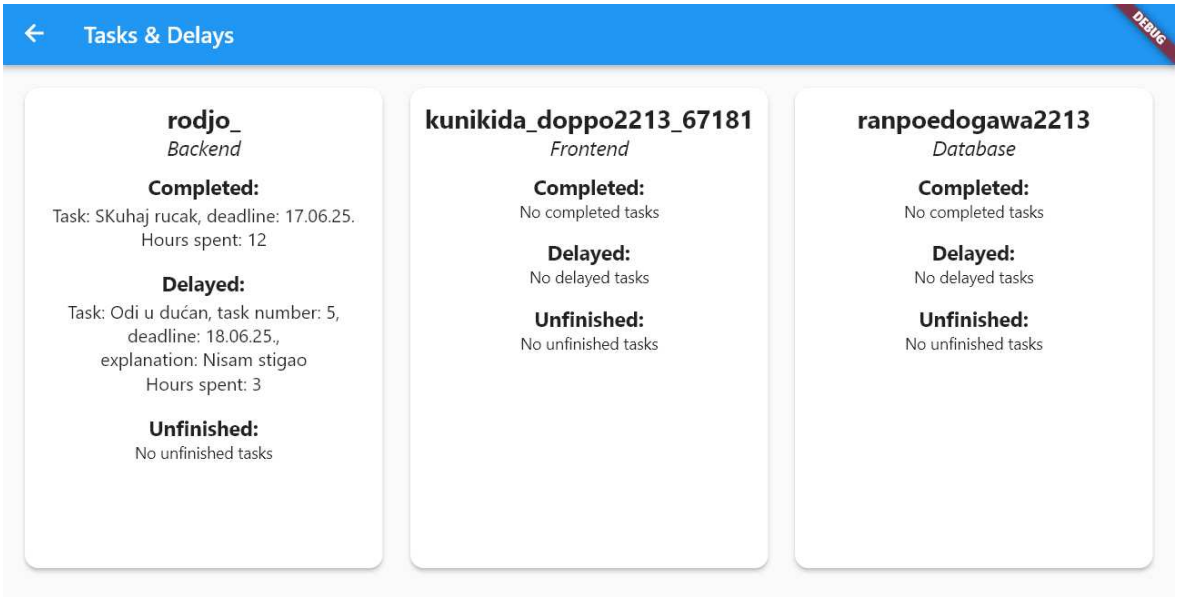
Na stranici *progress* sortira se datoteka *progress.txt* po članovima tima te se podatci za prikaz spremaju u mapu. Konačan izgled stranice dan je na slici (Slika 5.2):



Slika 5.2: Stranica *Progress*

5.1.5. Prikaz zadataka i kašnjenja - Tasks & Delays

Za prikaz zadataka i kašnjenja na stranici *tasks&delays* potrebno je više datoteka: sve *workday* datoteke i sve *delays* datoteke. Za svakog člana tima navedenog u mapi *users* otvara se prvo *delay* datoteka ako postoji te se *task ids* iz te datoteke spremaju u skup. Zatim se otvara odgovarajuća *workday* datoteka ako postoji te se zadatci sortiraju u tri kategorije: završeni zadatci koji će imati 0 kao *task id*, zadatci s kašnjenjem za koje se provjerava postoji li *task id* u skupu *delayed*, i zadatci koji su trenutno nedovršeni. Uz svaki zadatak prikazano je i koliko je sati utrošeno na njih (Slika 5.3).



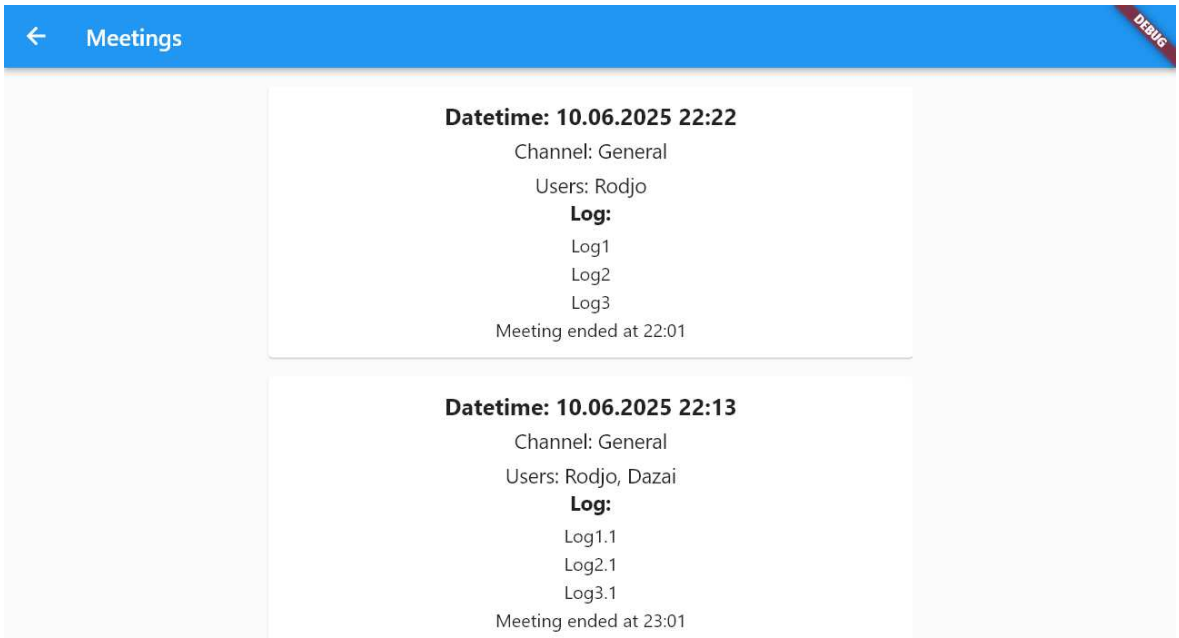
Slika 5.3: Stranica *tasks&delays*

5.1.6. Prikaz sastanaka - Meetings

Stranica *meetings* zadužena je za prikaz zapisnika sastanaka. Ona parsira datoteku *meetings.txt* te svaki sastanak sprema kao novi objekt:

```
class Meeting{
  final String datetime;
  final String channel;
  final List<String> users;
  final List<String> log;
}
```

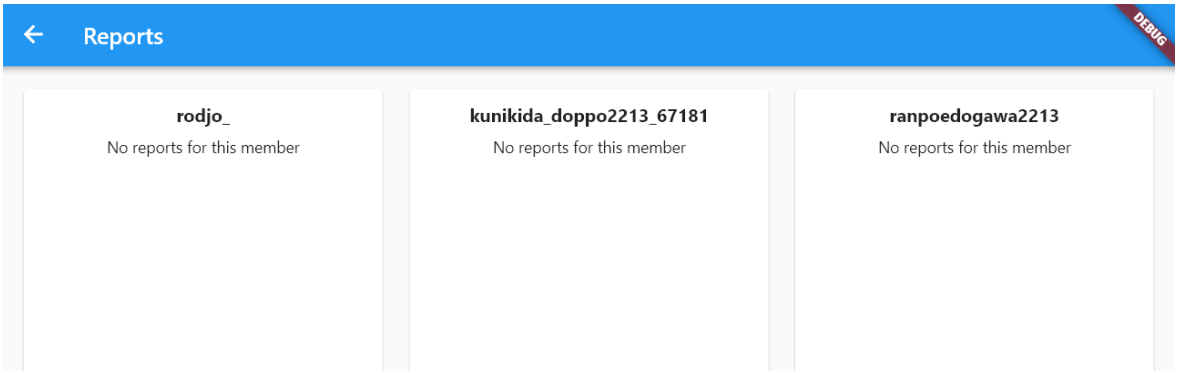
U datoteci, svaki sastanak počinje vremenom i datumom sastanka te se tako prepoznaje kad kreće zapisnik novog sastanka. Za svaki sastanak prikazuju se prvo datum i vrijeme sastanka, zatim kanal u kojem je proveden, članovi tima koji su mu prisustvovali te konačno sami zapisnik. Na dnu se ispisuje kada je sastanak završio (Slika 5.4).



Slika 5.4: Stranica *meetings*

5.1.7. Prikaz prijava - Reports

Stranica *reports* sortira datoteku *reports.txt* po članovima tima te prikazuje razloge prijave, ako je bilo prijava (Slika 5.5).

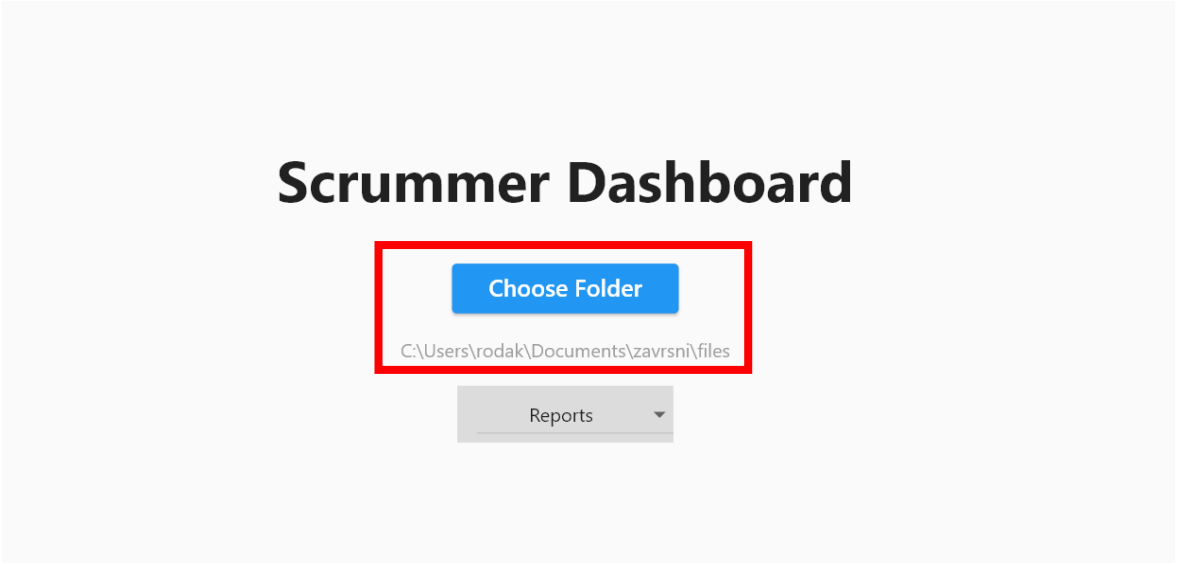


Slika 5.5: Stranica *reports*

5.2. Korištenje

Ako se želi koristiti ovakav *dashboard*, bez preinaka, dovoljno je s GitHub repozitorija samo preuzeti komprimiranu mapu *dashboard.zip* u kojemu se nalazi .exe datoteka. *Dashboard* je moguće mijenjati i prilagoditi svojim potrebama, ali u tom slučaju potrebno je preuzeti čitav izvorni kôd. Kôd se pokreće preko komandne linije ili preko terminala unutar uređivača teksta *VSCode* naredbom `flutter run`.

Pokretanjem programa otvorit će se početna stranica na kojoj je potrebno odabrati direktorij s datotekama. Kad se odabere, ili ako postoji već spremljeni direktorij, ispisat će se ispod gumba (Slika 5.6). Nakon odabira direktorija, bira se kategorija za koju se žele vidjeti informacije (Slika 5.7).



Slika 5.6: Početna stranica

Slika 5.7: Odabir kategorije

Na svakoj stranici, u gornjem lijevom kutu, nalazi se gumb koji preusmjerava natrag na početnu stranicu te se onda može odabrati nova kategorija.

6. Osvrt na ostvareno rješenje

Ovaj alat izgrađen je na temelju osobnog iskustva uspješnog vođenja projekata na fakultetu i iskustava drugih ljudi. Glavno pitanje koje se postavljalo za vrijeme implementacije jest: „Što bi bilo korisno imati?“ Odgovaranjem na to pitanje došlo se do prethodno opisanog izbora naredbi i funkcionalnosti.

Sve funkcionalnosti testirane su te sve rade kakve jesu (*as is, out of the box*). Međutim, zbog vremenskih ograničenja, rješenje se nije stiglo testirati na nekom pravom projektu. To znači da se, nažalost, može samo reći da bi „trebalo raditi“ i biti korisno, bez stvarnih dokaza i prakse. Ovo je jedan od razloga zašto je cijeli projekt stavljen na GitHub otkuda ga mogu preuzeti zainteresirani pojedinci i timovi te prilagoditi svojim potrebama ili dati povratnu informaciju.

Mjesta za doradu sigurno ima. *Dashboard* trenutno samo prikazuje gotove podatke iz datoteka, u budućnosti bi mu se moglo dodati generiranje grafova, vremenskih crta (*timeline*) itd. *Discord bot* također ima mjesta na kojima bi ga se moglo dodatno izbrusiti.

Jedno pitanje koje se postavilo krajem ovog projekta: „Je li moguće ovakvo nešto napraviti umjetnom inteligencijom?“ Odgovor na to je jedno veliko „Ne!“ Format datoteka u *botu*, format naredbi, izbor naredbi i procesuiranje podataka iziskivali su ljudsku kreativu. Umjetna inteligencija pomogla je pri formatiranju kôda i implementacijskih pitanja, ali općenitu razradu ideje i arhitekturu sustava umjetna inteligencija nije u stanju dati. Umjetna inteligencija eventualno bi dala nekakvo polovično, loše posloženo rješenje koje nije modularno, skalabilno ni lako održivo. Ljudski mozak bio je potreban kako bi se datoteke i naredbe formatirale na logičan i, koliko je to bilo moguće, jednostavan način. Što se tiče *dashboarda*, umjetna inteligencija može pomoći kod pisanja kôda za prikaz podataka, ali sami izgled mora osmisliti čovjek. Isto vrijedi i za procesuiranje podataka i parsiranje datoteka.

Ovaj projekt bio je prilika za dublje upoznavanje mogućnosti programskog jezika *Python*, alata *Flutter* i platforme *Discord*. *Python* je odabran jer je to standardan jezik za razvoj *botova*, uz veliku podršku zajednice i službeno programsko sučelje. Pokazao se kao odličan odabir, iako je samo programiranje *bota* dosta repetitivno.

Flutter se također pokazao kao odličan odabir. Lako se programira u njemu, kôd je pregledan i lako razumljiv te ga je lako *debuggirati*. Također, iznimno je lako u njemu razviti vizualno lijepo rješenje.

Zaključak

Na žalost svih programera, sastanci su neizbježan dio rada u poslovnom okruženju. Cilj i nastojanje ovoga rada bilo je razviti alat koji bi pomogao skratiti te sastanke na najmanju moguću mjeru, u svrhu povećanja produktivnosti, motivacije i kvalitete rada, uz pomoć jedne neuobičajene platforme. Platforma *Discord*, prvenstveno razvijena za *gamere*, pronašla je svoju upotrebu i u profesionalnom svijetu. Besplatna platforma, koja se može koristiti na računalu, na mobitelu i preko *web browsera*, omogućuje prilagodljivost za sve oblike primjene. Preko web stranice *Discord Developer* kreira se *bot* u nekoliko klikova, a daljnje programiranje svodi se na definiranje naredbi u Pythonu, uz pomoć službenog programskog sučelja *Discord API.* Korištenje *bota* svodi se na slanje naredbi u *Discord server* što omogućuje prikupljanje podataka o napretku projekta, a *dashboard* razvijen u alatu *Flutter* služi za pregledan prikaz tih podataka. Implementirane naredbe su one koje se mogu očekivati za ovu primjenu: naredba za prijavu napretka, naredba za prijavu zadataka koji se trebaju napraviti, naredba za prijavu kašnjenja, uz još nekoliko naredbi za poboljšanje kvalitete rada. Cijeli projekt objavljen je na *Githubu* otkuda ga zainteresirani timovi mogu preuzeti i prilagoditi svojim potrebama i željama.

Literatura

[1] Sruk V., *Modeli procesa programskog inžinjerstva,* 2024, listopad. Poveznica: https://moodle.fer.hr/pluginfile.php/81578/mod_resource/content/2/PROINZ_04_PROC_24.pdf; pristupljeno 12. lipnja 2025.

[2] Anonymous, *Waterfall Model – Software Engineering*, 2025., travanj. Poveznica: https://www.geeksforgeeks.org/software-engineering/waterfall-model/; pristupljeno 12. lipnja 2025.

[3] Anonymous, *Discord for Developers.* Poveznica: https://discord.com/developers; pristupljeno 12. lipnja 2025.

[4] Anonymous, *What is Scrum?*. Poveznica: https://www.scrum.org/resources/what-scrum-module; pristupljeno 12. lipnja 2025.

[5] Edwards J., *Why Does Scrum Make Programmers Hate Coding?.* Poveznica: https://www.linkedin.com/pulse/why-does-scrum-make-programmers-hate-coding-jayme-edwards/; pristupljeno 12. lipnja 2025.

[6] Anonymous, *About Discord | Our Mission and Story*. Poveznica: https://discord.com/company; pristupljeno 12. lipnja 2025.

[7] Albergotti R., *Facebook to Pay $19 Billion for WhatsApp,* 2014., travanj. Poveznica: https://www.wsj.com/articles/facebook-to-buy-whatsapp-for-16-billion-1392847766; pristupljeno 20. lipnja 2025.

[8] Olson P., *Exclusive: The Rags-To-Riches Tale Of How Jan Koum Built WhatsApp Into Facebook's New $19 Billion Baby,* 2014., veljača. Poveznica: https://www.forbes.com/sites/parmyolson/2014/02/19/exclusive-inside-story-how-jan-koum-built-whatsapp-into-facebooks-new-19-billion-baby/; pristupljeno 20. lipnja 2025.

[9] Anonymous, *Osjećajte se kao da ste u prostoriji sa svojim najbližima*. Poveznica: https://www.whatsapp.com/stayconnected; pristupljeno 20. lipnja 2025.

[10] Neate R., *Zoom booms as demand for video-conferencing tech grows*, 2020., ožujak. Poveznica: https://www.theguardian.com/technology/2020/mar/31/zoom-booms-as-demand-for-video-conferencing-tech-grows-in-coronavirus-outbreak; pristupljeno 20. lipnja 2025.

[11] Anonymous, *Plans & Pricing for Zoom Workplace.* Poveznica: https://zoom.us/pricing; pristupljeno 20. lipnja 2025.

[12] Warren T., *Microsoft Teams launches to take on Slack in the workplace,* 2016., studeni. Poveznica: https://www.theverge.com/2016/11/2/13497992/microsoft-teams-slack-competitor-features; pristupljeno 20. lipnja 2025.

[13] John W., *'What is Slack?' Everything you need to know about the professional messaging program.*, 2021., veljača. Poveznica: https://www.businessinsider.com/guides/tech/what-is-slack; pristupljeno 20. lipnja 2025.

[14] Tam D., *Flickr founder plans to kill company e-mails with Slack*, 2013., kolovoz. Poveznica: https://www.cnet.com/tech/tech-industry/flickr-founder-plans-to-kill-company-e-mails-with-slack/; pristupljeno 20. lipnja 2025.

[15] Anonymous, *Iterative vs Incremental model in Software Development*, 2023., prosinac. Poveznica: https://www.geeksforgeeks.org/software-engineering/iterative-vs-incremental-model-in-software-development/; pristupljeno 20. lipnja 2025.

Sažetak

Dodatak za aplikaciju Discord namijenjen učinkovitoj pripremi sastanaka u radnom okviru Scrum

Cilj ovoga rada bio je razviti alat kojim bi se dnevni sastanci u Scrum modelu skratili na najmanju moguću mjeru, s ciljem povećanja produktivnosti. Odabrana je platforma *Discord* zbog svoje izuzetno velike prilagodljivosti, jednostavnosti korištenja i činjenice da je za sve korisnike besplatna. Službeno programsko sučelje *Discord API* omogućava stvaranje vanjskih skripti (*botovi*) koje se spajaju na samu platformu, omogućujući beskonačnu prilagodljivost.

Programiranje *botova* svodi se na pisanje *Python* skripte. Broj naredbi, kao i njihova funkcionalnost, nije ograničen te se prepušta mašti programera i potrebama *servera* za koji se razvija. Korištenje samog *bota* svodi se na slanje poruka u *server*, pazeći da poruke počinju odgovarajućim prefiksom.

Bot koji je razvijen u sklopu ovoga projekta sadrži naredbe za prijavu poslova, zadataka, kašnjenja, napretka, članova tima, uz naredbe za zapisivanje sastanaka (*log*), glasanje (*poll*) i slanje datoteka. Podatci skupljeni preko *bota* spremaju se u tekstualne datoteke.

Radi preglednog prikaza podataka razvijen je *dashboard* u alatu *Flutter*. On čita i sortira datoteke po članovima tima te ih prikazuje po odabranim kategorijama.

Scrummer, Discord bot, Scrum, agile, Flutter, dashboard, sastanak, sastanci

Summary

A Discord plugin for efficient Scrum meeting preparation

Aim of this thesis was to develop a tool to shorten the Scrum daily standups as much as possible, with the aim of increasing productivity. Platform *Discord* was chosen for its exceptional customisation features, ease of use and the fact that it's free for all its users. Official application interface *Discord API* enables creation of external scripts (called *bots*) which connect to the platform itself, opening up endless opportunities.

Programming of *bots* trickles down to writing a *Python* script. The number of commands, as well as their functionalities, is not limited and is left to the imagination of the programmer and the needs of the *server* it's being developed for. Using the *bot* itself is essentially just sending messages to the *server,* with a correct prefix.

Bot developed in this project contains commands for reporting chores, tasks, delays, progress, team members, along with commands for creating meeting logs, creating polls and file sending. Data collected with the *bot* is saved in text files.

For displaying said data cleanly, a dashboard was developed in the *Flutter* framework. Dashboard parses the files and sorts the data by team members, then it displays them according to the selected categories.

Scrummer, Discord bot, Scrum, agile, Flutter, dashboard, meetings